# Кондиционеры версия 2

# Логика, которая касается пользователей.

## 03. 08. 2011

На данный момент приложение умеет содавать пользователей и назначать им роли –

|  |
| --- |
| ROLE\_USER, ROLE\_ADMIN, ROLE\_MANAGER, ROLE\_WORKER |

У каждой роли есть свои права.

|  |
| --- |
| READ, WRITE, DELETE |

|  |  |
| --- | --- |
| RoleEntity workerRole = createRole(Roles.ROLE\_WORKER.name(), Arrays.asList(  readAuthority)); | ROLE\_WORKER |
| RoleEntity adminRole = createRole(Roles.ROLE\_ADMIN.name(), Arrays.asList(readAuthority,  writeAuthority, deleteAuthority)); | ROLE\_ADMIN |
| RoleEntity managerRole = createRole(Roles.ROLE\_MANAGER.name(), Arrays.asList(readAuthority,  writeAuthority)); | ROLE\_MANAGER, |

Кроме этого, есть контроллер AdminUserController с методами –

|  |  |
| --- | --- |
| createUser | ROLE\_ADMIN |

Основные задачи дальше – доделать админку.

|  |  |  |
| --- | --- | --- |
| 1. изменить пользователя | ROLE\_ADMIN  ROLE\_MANAGER  ROLE\_WORKER(только себя) | Контроллер управления  Профайл |
| 1. добавить роль конкретному пользователю | ROLE\_ADMIN | Админка |
| 1. Удалить роль у конкретного пользователя | ROLE\_ADMIN | Админка |
| 1. получить пользователя | ROLE\_ADMIN  ROLE\_MANAGER  ROLE\_WORKER(только себя) | Контроллер управления  Админка  Профайл |
| 1. получить всех пользователей | ROLE\_ADMIN  ROLE\_MANAGER | Контроллер управления |
| 1. удалить пользователя | ROLE\_ADMIN | Админка |
| 1. заблокировать пользователя | ROLE\_ADMIN | Админка |

## 03. 08. 2011 - 2

На данный момент в админке созданы методы -

|  |  |  |
| --- | --- | --- |
| 1. удалить пользователя | ROLE\_ADMIN | Админка |
| 1. заблокировать пользователя | ROLE\_ADMIN | Админка |

Кроме этого, есть контроллер AdminUserController с методами –

|  |  |
| --- | --- |
| createUser | ROLE\_ADMIN |
| deleteUser | ROLE\_ADMIN |
| blockUser | ROLE\_ADMIN |

## 04.08.2021

На данный момент в админке есть методы –

|  |  |
| --- | --- |
| createUser | ROLE\_ADMIN |
| deleteUser | ROLE\_ADMIN |
| blockUser | ROLE\_ADMIN |
| addRoleToUser | ROLE\_ADMIN |
| deleteRoleFromUser | ROLE\_ADMIN |

Следующая задача – создать контроллеры для менеджера и работника

## 05.08.2021

Созданы методы обновления пользователя и получения пользователя. Оба эти метода должны быть доступным всем трем ролям, но с разницей, что точка входа для них будет разная

На данный момент функциональность реализована для менеджера и создан менеджер контроллер.

На данный момент в админке есть методы –

|  |  |
| --- | --- |
| createUser | ROLE\_ADMIN |
| deleteUser | ROLE\_ADMIN |
| blockUser | ROLE\_ADMIN |
| addRoleToUser | ROLE\_ADMIN |
| deleteRoleFromUser | ROLE\_ADMIN |
|  |  |

На данный момент в общем сайте для менеджеров есть методы –

|  |  |
| --- | --- |
| updateUser | ROLE\_ADMIN |
| getUserByUuid | ROLE\_ADMIN |

Следующий шаг – добавить эти же два метода (обновление пользователя и получение пользователя) для администратора

На данный момент в админке есть методы –

|  |  |
| --- | --- |
| createUser | ROLE\_ADMIN |
| deleteUser | ROLE\_ADMIN |
| blockUser | ROLE\_ADMIN |
| addRoleToUser | ROLE\_ADMIN |
| deleteRoleFromUser | ROLE\_ADMIN |
| updateUser | ROLE\_ADMIN |
| getUserByUuid | ROLE\_ADMIN |

На данный момент в общем сайте для менеджеров есть методы –

|  |  |
| --- | --- |
| updateUser | ROLE\_MANAGER |
| getUserByUuid | ROLE\_MANAGER |

Следующий шаг – создать метод getAllUsers для администратора и менеджера. И добавить profileController с методами обновления и получения пользователя для работника

На данный момент в админке есть методы –

|  |  |
| --- | --- |
| createUser | ROLE\_ADMIN |
| deleteUser | ROLE\_ADMIN |
| blockUser | ROLE\_ADMIN |
| addRoleToUser | ROLE\_ADMIN |
| deleteRoleFromUser | ROLE\_ADMIN |
| updateUser | ROLE\_ADMIN |
| getUserByUuid | ROLE\_ADMIN |
| getAllUsers | ROLE\_ADMIN |

На данный момент в общем сайте для менеджеров есть методы –

|  |  |
| --- | --- |
| updateUser | ROLE\_MANAGER |
| getUserByUuid | ROLE\_MANAGER |
| getAllUsers | ROLE\_MANAGER |

На данный момент в profile сайте для работников есть методы – эти методы разрешены пользователю исключительно в отношении себя самого

|  |  |
| --- | --- |
| updateUser | ROLE\_WORKER |
| getUserByUuid | ROLE\_WORKER |

!!! метод «Обновить пользователя» позволяет менять имя, фамилию и электронный адрес пользователя

На данном этапе эта логика закончена.

Следующий шаг – тесты и документация

### Документация

На данном этапе создается документация посредством javaDoc. Для ее создания необходимо ввести команду mvn javadoc:javadoc и документация будет создана в корневой папке проекта

### Тестирование

Готовы тесты Админ сервиса

Готовы тесты GeneralService

06.08.2021

На данном этапе все методы и контроллеры, связанные с пользователем написаны. Все методы сервиса покрыты тестами

На данный момент в админке есть методы –

|  |  |
| --- | --- |
| createUser | ROLE\_ADMIN |
| deleteUser | ROLE\_ADMIN |
| blockUser | ROLE\_ADMIN |
| addRoleToUser | ROLE\_ADMIN |
| deleteRoleFromUser | ROLE\_ADMIN |
| updateUser | ROLE\_ADMIN |
| getUserByUuid | ROLE\_ADMIN |
| getAllUsers | ROLE\_ADMIN |

На данный момент в общем сайте для менеджеров есть методы –

|  |  |
| --- | --- |
| updateUser | ROLE\_MANAGER |
| getUserByUuid | ROLE\_MANAGER |
| getAllUsers | ROLE\_MANAGER |

На данный момент в profile сайте для работников есть методы – эти методы разрешены пользователю исключительно в отношении себя самого

|  |  |
| --- | --- |
| updateUser | ROLE\_WORKER |
| getUserByUuid | ROLE\_WORKER |

Следующая задача – подключить Spring Security так, что бы открыть доступ к конкретным методам только тем, кто конкретно имеет к ним доступ.

Ветка -> Security

После того, как мы добавляем зависимость в pom - при запросе из постмена мы получаем ошибку 401 - Unauthorized. Пока все работает как нужно

После добавления метода

@Override  
protected void configure(HttpSecurity http) throws Exception {  
 http.csrf().disable()  
 .authorizeRequests()  
 .anyRequest().authenticated();  
  
 http.headers().frameOptions().disable();  
}

В класс

public class WebSecurity extends WebSecurityConfigurerAdapter

мы должны в постмене получить уже 403 ошибку.

Эта ошибка говорит о том, что нам необходимо сначала пройти аутентификацию (то есть доказать, что у нас есть право доступа к этому методу) и потом только контроллер нас в него пропустит.

Для того, что бы пройти – нам необходим токен.

Добавляем создание и обработку токена

### Идея токена

Идея в том, что при логине пользователь получит свой токен, по которому его будет определять юзер сервис и давать ему доступ ко всему

### Создание токена

Создаем класс LoginRequestDto. Именно объект этого класса должен отправить пользователь при «логине»

После этого необходимо создать класс, который, по сути, будет являться фильтром аутентификации и который будет создавать JWT token.

public class AutheficationFilter extends UsernamePasswordAuthenticationFilter

В этом классе нам необходимо создать несколько методов –

1. Метод, который будет вызываться каждый раз, когда пользователь будет отправлять запрос по адресу {url}/login

@Override  
public Authentication attemptAuthentication(HttpServletRequest request, HttpServletResponse response)

После того, как метод получит пользователя (пользователь приходит из HttpServletRequest request, модель пользователя - LoginRequestDto ), он вызывает метод loadUserByUsername. Это метод, который физически проверяет, наличие пользователя в БД и то, совпадает ли пользователь с те, который пришел.

1. Создание метода loadUserByUserName()

По сути, это метод, который переопределяется из класса USerDetailService (класс спринга), поэтому для его создания и переопределения необходимо создать сервис, который будет наследовать этот класс.

Создаем интерфейс UserSecurityService, в котором будет метод – loadUserByUserName().

@Override  
public UserPrincipal loadUserByUsername(String email) throws UsernameNotFoundException {  
 UserEntity user = userRepository.findUserByUserEmail(email);  
 if (user == null) {  
 throw new UsernameNotFoundException(LoggerConstants.*USER\_WITH\_EMAIL* +  
 email + LoggerConstants.*NOT\_FOUND\_IN\_DB*);  
 }  
 return new UserPrincipal(user);  
}

В этом методе мы возвращаем Principal. И для того, что бы вернут именно те данные о принципале, которые мы хотим – необходимо переопределить этот класс.

1. Т е создаем еще и класс UserPrincipal.

public class UserPrincipal implements UserDetails

1. Если пользователь получен из базы данных и с ним все ОК – вызывается второй метод из класса AutheficationFilter это метод

protected void successfulAuthentication(HttpServletRequest request,  
 HttpServletResponse response,  
 FilterChain chain,  
 Authentication authResult)

И уже этот метод должен сгенерировать Token и добавить его в Header для ответа.

Кроме этого был создан класс

public class SecurityConstants

в котором будут храниться все константы, связанные с безопасностью.

И для того, что бы TokenSecret, который добавлен в applicationProErties был доступен из контекста, его нужно туда добавить.

public class AppProperties {  
  
 @Autowired  
 private Environment env;  
  
 public String getTokenSecret()  
 {  
 return env.getProperty("tokenSecret");  
 }  
}

Так же был добавлен в класс BeanConfig – Bean для SpringApplicationContext

@Bean  
public SpringApplicationContext springApplicationContext() {  
 return new SpringApplicationContext();  
}

@Bean(name = "AppProperties")  
public AppProperties getAppProperties() {  
 return new AppProperties();  
}

Для того, что бы дать классу UserServiceImpl доступ к ApplicationContext –

public class SpringApplicationContext implements ApplicationContextAware {  
 private static ApplicationContext *CONTEXT*;  
 @Override  
 public void setApplicationContext(ApplicationContext context) throws BeansException {  
 *CONTEXT* = context;  
 }  
 public static Object getBean(String beanName) {  
 return *CONTEXT*.getBean(beanName);  
 }

Добавляем все это в бины –

@Component  
public class BeanConfig {  
 */\*\*  
 \* bean that create BCryptPasswordEncoder Bean  
 \*  
 \** ***@return*** *new {****@link*** *BCryptPasswordEncoder}  
 \*/* @Bean  
 public BCryptPasswordEncoder bCryptPasswordEncoder() {  
 return new BCryptPasswordEncoder();  
 }  
  
 @Bean  
 public SpringApplicationContext springApplicationContext() {  
 return new SpringApplicationContext();  
 }  
  
 @Bean(name = "AppProperties")  
 public AppProperties getAppProperties() {  
 return new AppProperties();  
 }  
}

И наконец последнее что нужно сделать – это добавить фильтр в WebSecurity

Для этого в классе создается метод

protected void configure(AuthenticationManagerBuilder auth) throws Exception

который будет использовать фильтр attemptAuthentication каждый раз, когда нужно будет расшифровать пароль из БД

и метод

protected AuthenticationFilter getAuthenticationFilter() throws Exception

в котором сразу же меняем точку входа для логина – на {url}/users/login

Теперь если запустить приложение и послать запрос – мы должны получить токен

# 07.08.2011

Следующий шаг – создание фильтра, который должен проверить – релевантный ли токен у пользователя при доступах к методам.

Пока мы все еще будем получать 403 ошибку в методах, потому что мы не написали функционал, который будет проверять токен

## Добавляем авторизацию

Делаем так, что бы только авторизованные пользователи могли получить доступ к приложению user-service. Используем токен для проверки авторизации

Каждый раз, когда происзодит обращение к любой конечной точке, которая требует авторизации, происходит вызов метода doFilterInternal

Делаем так, что бы только авторизованные пользователи могли получить доступ к приложению user-service. Используем токен для проверки авторизации

1. Пишем класс AuthorizationFilter

public class AuthorizationFilter extends BasicAuthenticationFilter {  
   
 public AuthorizationFilter(AuthenticationManager authenticationManager) {  
 super(authenticationManager);  
 }  
  
}

1. Каждый раз, когда происзодит обращение к любой конечной точке, которая требует авторизации, происходит вызов метода doFilterInternal

Метод имеет доступ к headers

@Override  
protected void doFilterInternal(HttpServletRequest request, HttpServletResponse response,  
 FilterChain chain) throws IOException, ServletException {  
 String header = request.getHeader(SecurityConstants.*HEADER\_STRING*);  
 if(header == null || !header.startsWith(SecurityConstants.*TOKEN\_PREFIX*)){  
 chain.doFilter(request, response);  
 return;  
 }  
 UsernamePasswordAuthenticationToken authentication = getAuthentication(request);  
 SecurityContextHolder.*getContext*().setAuthentication(authentication);  
 chain.doFilter(request, response);  
}

1. Когда метод проверил что все ОК, и токен в принципе присутствует, он вызыввает метод getAuthentication который проверяет релевантность токена

private UsernamePasswordAuthenticationToken getAuthentication(HttpServletRequest request) {  
 String token = request.getHeader(SecurityConstants.*HEADER\_STRING*);  
  
 if (token != null) {  
 token = token.replace(SecurityConstants.*TOKEN\_PREFIX*, "");  
 String user = Jwts.*parser*()  
 .setSigningKey(SecurityConstants.*getTokenSecret*())  
 .parseClaimsJws(token)  
 .getBody()  
 .getSubject();  
 if (user != null) {  
 return new UsernamePasswordAuthenticationToken(user,  
 null, new ArrayList<>());  
 }  
 return null;  
 }  
 return null;  
}

1. Добавляем фильтр в конфигурацию WebSecurity

.and()  
 .addFilter(getAuthenticationFilter())  
 */\*\*  
 \* checking authorization  
 \*/* .addFilter(new AuthorizationFilter(authenticationManager()));  
  
*/\*\*  
 \* clean headers after all request*

1. Кроме этого, добавляем настройки CORS

*/\*\**

*\* add cors  
 \*/*@Bean  
public CorsConfigurationSource corsConfigurationSource() {  
 final CorsConfiguration configuration = new CorsConfiguration();  
  
 configuration.setAllowedOrigins(Arrays.*asList*("\*"));  
 configuration.setAllowedMethods(Arrays.*asList*("GET", "POST", "PUT", "DELETE", "OPTIONS"));  
 configuration.setAllowCredentials(true);  
 configuration.setAllowedHeaders(Arrays.*asList*("\*"));  
  
 final UrlBasedCorsConfigurationSource source = new UrlBasedCorsConfigurationSource();  
 source.registerCorsConfiguration("/\*\*", configuration);  
  
 return source;  
}

Теперь у приложения есть определенные методы, и кроме этого, есть api login который доступен всем, и api который доступны исключительно авторизированным пользователям

Осталось сделать доступ по ролям и протестировать все

# Реализация фильтров для проверки роли и прав

Теперь когда пользователь заходит в систему, нам нужно проверять какая у него роль и права, и установить их для этого пользователя. По сути, теперь наш токен должен содержать еще и роль пользователя. Заодно добавим в header имя пользователя, его email и роль.

Сейчас во время логина у нас срабатывает Authentication filtr. Этот метод вызывает loadUserByName(), в котором возвращает пользователя по заданным параметрам. Теперь нужно, что бы этот метод добавлял еще и authorities возвращаемому пользователю.